```
#include "GLee/GLee.h"
#include <GL/glut.h>
#include <iostream>
#include <stdio.h>
#include <windows.h>
#include "time.h"
#include "Maths/Maths.h"
#include "Animation.h"

using namespace std;

GLuint shadow_map; // to store the shadow

unsigned width, height; // of the window
unsigned shadow_size = 512; // size of the shadow
unsigned scene = 1; // current scene
float position; // position from the timer
float pos; // actual position
float rotation; // rotation

MATRIX4X4 bias(0.5f, 0.0f, 0.0f, 0.0f,
       0.0f, 0.5f, 0.0f, 0.0f,
       0.0f, 0.0f, 0.5f, 0.0f,
       0.5f, 0.5f, 0.5f, 1.0f); // for the matrix calculation

MATRIX4X4 projection_camera, projection_light, view_camera, view_light;
// matricies for the camera and the light

VECTOR3D light_pos(2.0f, 3.0f, -2.0f);
VECTOR3D camera_pos(-2.5f, 3.5f, -2.5f);
// position of the light and camera

time timer;
// a timer

void Display(void)
{
       drawFromLight(); // draw from light point of view
       drawFromCamera(); // draw from camera point of view
       renderShadow(); // calculate shadow

       glutSwapBuffers();
       glutPostRedisplay();

}

void drawFromLight()
{
       position = timer.currentTime() / 10; // get the current time in the scene

       glLoadIdentity();
       gluLookAt(light_pos.x, light_pos.y, light_pos.z,
              0.0f, 0.0f, 0.0f,
              0.0f, 1.0f, 0.0f); // move to the light
       glGetFloatv(GL_MODELVIEW_MATRIX, view_light); // store the light view
       glPopMatrix();

       if (scene == 10)
```

```
    {
        light_pos.z = light_pos.z + position / 50000;
        // move the light on z if on scene 10
    }

    else if (scene == 11)
    {
        light_pos.z = 2;
        light_pos.x = light_pos.x - position / 50000;
        // move the light on x if on scene 11
    }


    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);
    glLoadMatrixf(projection_light);
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(view_light);
    // load the light matrices

    glColorMask(0, 0, 0, 0);
    glCullFace(GL_FRONT);
    glViewport(0, 0, shadow_size, shadow_size);


    drawObjects();
    //render objects

    glBindTexture(GL_TEXTURE_2D, shadow_map);
    glCopyTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 0, 0, shadow_size, shadow_size);

    glCullFace(GL_BACK);
    glColorMask(1, 1, 1, 1);
}

void drawFromCamera()
{
    glClear(GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);
    glLoadMatrixf(projection_camera);
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(view_camera);
    // load camera matrices

    glViewport(0, 0, width, height);

    glLightfv(GL_LIGHT0, GL_SPECULAR, black);
    glLightfv(GL_LIGHT0, GL_AMBIENT, black);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, black);
    glLightfv(GL_LIGHT0, GL_POSITION, VECTOR4D(light_pos));
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);
    // turn on the lights but make them black

    drawObjects();
}
```

```
void renderShadow()
{
	glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
	glLightfv(GL_LIGHT0, GL_SPECULAR, white);
	// make lights white

	MATRIX4X4 texture = bias*projection_light*view_light;
	// calculate shadow map

	glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
	glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
	glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
	glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
	glTexGenfv(GL_S, GL_EYE_PLANE, texture.GetRow(0));
	glTexGenfv(GL_T, GL_EYE_PLANE, texture.GetRow(1));
	glTexGenfv(GL_R, GL_EYE_PLANE, texture.GetRow(2));
	glTexGenfv(GL_Q, GL_EYE_PLANE, texture.GetRow(3));
	glEnable(GL_TEXTURE_GEN_S);
	glEnable(GL_TEXTURE_GEN_T);
	glEnable(GL_TEXTURE_GEN_R);
	glEnable(GL_TEXTURE_GEN_Q);
	glBindTexture(GL_TEXTURE_2D, shadow_map);
	glEnable(GL_TEXTURE_2D);
	glTexParameteri(GL_TEXTURE_2D, GL_DEPTH_TEXTURE_MODE, GL_INTENSITY);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT24, shadow_size, shadow_size, 0,
GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);
	// generate shadow map

	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE, GL_COMPARE_R_TO_TEXTURE);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
	glAlphaFunc(GL_GEQUAL, 0.99f);
	glEnable(GL_ALPHA_TEST);
	// work out what is in the shadow

	drawObjects();
	// draw objects again

	glDisable(GL_TEXTURE_2D);
}

void drawObjects()
{
	pos = position / 500;
	rotation = position / 2;
	// get pos and rot vectors from current time

	glColor3f(1.0f, 1.0f, 1.0f);
	glPushMatrix();
	glScalef(1.0f, 0.05f, 1.0f);
	glutSolidCube(3.0f);
	glPopMatrix();
	// draw the plane

	if (scene < 10)
	{
		VECTOR3D lightPosition(2.0f, 3.0f, -2.0f);
```

```
    glColor3f(1.0f, 1.0f, 0.0f);
    glPushMatrix();
    glTranslatef(1.0f, 1.7f, -1.5f);
    glutSolidSphere(0.2, 24, 24);
    glPopMatrix();
    // draw a sphere for where the light is
}

if (scene == 1)
{
    if (pos > 0.75f)
    {
        changeScene();
        // when time is up change scene
    }

    glPushMatrix();
    glColor3f(1.0f, 0.0f, 0.0f);
    glTranslatef(0.0f, 0.5f, 0.0f);
    glutSolidCube(0.5);
    glPopMatrix();
    // draw a cube
}

else if (scene == 2)
{
    glPushMatrix();
    glColor3f(1.0f, 0.0f, 0.0f);

    if (pos <= 1.0f) // if the time of the scene is not up
    {
        glTranslatef(pos, 0.5f, 0.0f);
        // translate across X
    }

    else {
        glTranslatef(0.0f, 0.5f, 0.0f);
        changeScene();
        // if time is up move to next scene
    }

    glutSolidCube(0.5);
    glPopMatrix();
}

else if (scene == 3)
{
    glPushMatrix();
    glColor3f(1.0f, 0.0f, 0.0f);

    if (pos *-1 >= -2.3f)
    {
        glTranslatef(1 - pos, 0.5f, 0.0f);
        // translate back across X
    }

    else {
        glTranslatef(0.0f, 0.5f, 0.0f);
```

```
                changeScene();
        }

        glutSolidCube(0.5);
        glPopMatrix();
}

else if (scene == 4)
{
        glPushMatrix();
        glColor3f(1.0f, 0.0f, 0.0f);
        if (pos < 2.3f)
        {
                glTranslatef(1 - pos, 0.5f, -1.3 + pos);
                // translate across Z
        }

        else {
                glTranslatef(-1.3f, 0.5f, 1.0f);
                changeScene();
        }

        glutSolidCube(0.5);
        glPopMatrix();
}

else if (scene == 5)
{
        glPushMatrix();
        glColor3f(1.0f, 0.0f, 0.0f);

        if (pos < 2.3f) {
                glTranslatef(-1.3f + pos, 0.5f, 1.0f - pos);
                // translate across X and Z
        }

        else {
                glTranslatef(1.0f, 0.5f, -1.3f);
                changeScene();
        }

        glutSolidCube(0.5);
        glPopMatrix();
}

else if (scene == 6)
{
        glPushMatrix();
        glColor3f(1.0f, 0.0f, 0.0f);

        if (pos<= 2.3f)
        {
                glTranslatef(-1.3f + pos, 1.0f, -0.5f);
                // translate across X
        }

        else {
                glTranslatef(1.0f, 1.0f, -0.5f);
                changeScene();
        }

        glutSolidCube(0.5);
        glPopMatrix();

}

else if (scene == 7)
{
        glPushMatrix();
        glColor3f(1.0f, 0.0f, 0.0f);

        if (pos <= 2.0f)
        {
                glTranslatef(1.0f, 0.5f + pos, -1.0);
                // translate across Y
        }

        else {
                glTranslatef(1.0f, 1.5f, -1.0f);
                changeScene();
        }

        glutSolidCube(0.5);
        glPopMatrix();
}

else if (scene == 8)
{
        if (pos <= 4.0f)
        {
                glPushMatrix();
                glRotatef(rotation, 0.0f, 1.0f, 0.0f);
                // rotation across Y
                glColor3f(1.0f, 0.0f, 0.0f);
                glPushMatrix();

                glTranslatef(1.0f, 0.2f, 0.45f);
                glutSolidSphere(0.2, 24, 24);
                glPopMatrix();

                glPopMatrix();
        }

        else
        {
                changeScene();
        }
}

else if (scene == 9)
{
        if (pos <= 4.0f)
        {
                glPushMatrix();
                glRotatef(rotation, 0.0f, 1.0f, 0.0f);
```

```
            glColor3f(1.0f, 0.0f, 0.0f);
            glPushMatrix();
            glTranslatef(0.2f, 0.2f, 0.5f);
            glutSolidSphere(0.2, 24, 24);
            glPopMatrix();

            glPopMatrix();

            glPushMatrix();
            glRotatef(rotation * -1, 0.0f, 1.0f, 0.0f);

            glColor3f(1.0f, 0.0f, 0.0f);
            glPushMatrix();
            glTranslatef(0.2f, 1.0f, 0.5f);
            glutSolidSphere(0.2, 24, 24);
            glPopMatrix();

            glPopMatrix();
            // create two rotating spheres
            // at different Y positions

            glColor3f(0.0f, 1.0f, 0.0f);
            glPushMatrix();
            glTranslatef(0.0f, 0.5f, -0.05f);
            glScalef(1.0f, 5.0f, 1.0f);
            glutSolidCube(0.5);
            glPopMatrix();
        }

        else {
            changeScene();
        }
    }

    else if (scene == 10)
    {
        if (pos > 0.75f)
        {
            changeScene();
        }

        glColor3f(1.0f, 1.0f, 0.0f);
        glPushMatrix();

        glTranslatef(2.0f, 1.0f, -1.0f + position / 100);
        glutSolidSphere(0.25, 22, 22);
        glPopMatrix();
        // animate the sphere representing the light source

        glPushMatrix();
        glColor3f(1.0f, 0.0f, 0.0f);
        glTranslatef(0.0f, 0.5f, 0.0f);
        glutSolidCube(0.5);
        glPopMatrix();
    }

    else if (scene == 11)
    {
```

```
                if (pos > 0.75f)
                {
                        changeScene();
                }

                glColor3f(1.0f, 1.0f, 0.0f);
                glPushMatrix();
                glTranslatef(2.0f - position / 100, 1.0f, 2.0f);
                glutSolidSphere(0.25, 22, 22);
                glPopMatrix();
                // animate the sphere representing the light source

                glPushMatrix();
                glColor3f(1.0f, 0.0f, 0.0f);
                glTranslatef(0.0f, 0.5f, 0.0f);
                glutSolidCube(0.5);
                glPopMatrix();
        }
}


void changeScene() {
        light_pos.x = 2.0f;
        light_pos.z = -2.0f;
        timer.resetTime();
        position = timer.currentTime() / 10;
        scene = scene + 1;
        pos = position / 500;
        rotation = position / 2;

        if (scene == 12)
        {
                scene = 1;
                // if last scene return to first
        }
}

//Called for initiation
bool Init(void)
{
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
        glClearDepth(1.0f);
        glDepthFunc(GL_LEQUAL);
        glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_CULL_FACE);
        glEnable(GL_COLOR_MATERIAL);
        // setup

        glGenTextures(1, &shadow_map);
        glBindTexture(GL_TEXTURE_2D, shadow_map);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, shadow_size, shadow_size, 0,
                GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
        // init shadow map

        glPushMatrix();
        glLoadIdentity();
        gluPerspective(45.0f, (float)width / height, 1.0f, 100.0f);
        glGetFloatv(GL_MODELVIEW_MATRIX, projection_camera);
        glLoadIdentity();
        gluLookAt(camera_pos.x, camera_pos.y, camera_pos.z,
                0.0f, 0.0f, 0.0f,
                0.0f, 1.0f, 0.0f);
        glGetFloatv(GL_MODELVIEW_MATRIX, view_camera);
        glLoadIdentity();
        gluPerspective(45.0f, 1.0f, 2.0f, 8.0f);
        glGetFloatv(GL_MODELVIEW_MATRIX, projection_light);
        // save camera and light matricies
        return true;
}

int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitWindowSize(1000, 800);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGB);
        glutCreateWindow("Assignment");

        if (Init())
        {
                glutDisplayFunc(Display);
                glutReshapeFunc(Reshape);
                glutMainLoop();
        }
        return 0;
}

void Reshape(int w, int h)
{
        width = w, height = h;

        glPushMatrix();
        glLoadIdentity();
        gluPerspective(45.0f, (float)width / height, 1.0f, 100.0f);
        glGetFloatv(GL_MODELVIEW_MATRIX, projection_camera);
        glPopMatrix();
}
```